Leiden University

Computer Science and Economics

Improving and Evaluating AI Generated Feedback for Student Software Development

Name:	Daniel Tse
Student ID:	s3701131
Date:	15/07/2025
1st supervisor:	Joost Visser
2nd supervisor:	Kristoffer Kalavainen

BACHELOR'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Einsteinweg 55
2333 CC Leiden
The Netherlands

Abstract

Background: In software engineering education, students learn by developing software systems in teams. While detailed feedback on code, code organisation and other development aspects can help students improve, teachers and teaching assistants have limited time to provide such feedback. This study explores the use of a GenAI-driven tool, based on the work of Spinellis, to generate feedback on student's repositories. The tool was applied on a platform called LUDev where students could work on real world software engineering projects at Leiden University. Related works by Oliveira, Alyoshyna and Leong & Sung offer insights into automated peer review, student preferences and AI generated feedback across different domains. The studies highlight key challenges such as contextual relevance and prompt quality.

Aim: The goal of this research is to evaluate the effectiveness and actionability of AI-generated feedback in education, specifically in the context of student software engineering projects.

Method: An existing tool for feedback on Java projects was extended to Pyhton projects and applied to analyse the LUDev student repositories and produce automated feedback reports. The feedback reports were then distributed to the student teams together with a survey they were asked to complete. The survey was about the clarity, usefulness and actionability of the feedback. The results of the survey were analyzed to assess the success of the tool.

Results: The study shows that students found the feedback to be generally clear and moderately useful. The results highlight that feedback on source code and dependency management were most valued, while other areas received mixed responses.

Discussion: The results suggest that generative AI can support programming education by offering scalable and timely feedback. Nevertheless, some feedback points were too generic or missed important project-specific nuances. These findings highlight the importance of prompt quality.

Conclusion: The study investigated the use of a generative AI tool to provide automated feedback on software engineering projects from students. The findings indicate that AI generate feedback is a useful supplement to human feedback, with future improvements needed in areas such as detecting the programming detection, prompt design, continuous feedback, explainable AI techniques and comparing it with human feedback.

Contents

Chapter 1

Introduction

1.1 Problem statement

Artificial Intelligence (AI) is increasingly being applied across different areas of industry and research. In software engineering particular you can find AI in almost every step of the process. A common way that AI is used is through generating feedback. AI-generated feedback has the potential to help users identify issues and make improvements in their code.

Software engineers have already been utilizing this capability, which raises the question of whether this could also be useful in programming education. Lecturers and instructors face dual pressure from limited time and growing classes, while students require a detailed guidance on code organization, documentation and repository management [7]. These skills are increasingly important as modern software development revolves around collaborative workflows centered around version control systems like Git.

This thesis explores the application of AI-generated repository feedback as a solution to this educational challenge. This research aims to determine the effectiveness of AI generated feedback as a tool in programming education. The research builds on the AI-repo-feedback tool developed by Diomidis Spinellis [10] to generate feedback reports. To determine the effectivness of AI generated feedback in education, our research will be conducted on LUdev's [1] student project repositories. LUdev is a student run (virtual) platform which connect Leiden University students with companies to work on real world tech projects.

1.2 Research questions

Our research is structured around a set of research questions that guide this research process. The main research question concerns the fundamental motivation of the research: as class sizes grow and teaching resources remain limited, there is a need for scalable, and timely feedback on student code, this research investigates whether generative AI can fill that gap. The remaining supporting research questions answer different aspects of this research. Each supporting research question represents the main aim of one of the phases of the research.

Main research question RQ.0 How effective is the AI Repo Feedback tool in providing meaningful and actionable feedback on student software engineering projects?

Supporting RQ.1 How can the AI Repo Feedback tool be modified and extended to support Python-based repositories?

Supporting RQ.2 How useful and clear do LUDev students find AI-generated feedback and which aspect of the feedback report is considered most valuable?

Supporting RQ.3 a

1.3 Overview of the thesis

In Chapter 2, we discuss the related work and some necessary background information.

In Chapter 3, we discuss the methods used and the structure of this research.

In Chapter 4, we discuss the results from developing the tool and the evaluation survey.

In Chapter 5, we interpret the results in the context of our research questions. We do this by discussing implications of the findings and limitations of the research. Also suggest future research.

In Chapter 6, we finalize this research by answering the research questions and discussing some potential future work.

Chapter 2

Background and related work

2.1 Generative AI and API keys

Recent advances in Generative Artificial Intelligence (GenAI) have significantly impacted the field of software engineering. Generative AI is now being used for a variety of tasks in software engineering, such as code completion, documentation generation, error explanation and automated feedback. These capabilities help developers write cleaner and more maintainable code. From these capabilities came many tools that help developers use these functionalities. Tools such as GitHub Copilot [3] help developers receive contextual suggestions and feedback during their development process. These AI assistance tools go beyond just accelerating coding tasks; they also serve an educational purpose by providing suggestions and identifying potential issues [9].

2.2 DSpinellis AI-repo-feedback tool

The main tool that is used for this research was recently developed by Diomidis Spinellis and is called ai-repo-feedback [10]. It is an open-source tool that can automatically assess and provide structured feedback on Java repositories. The tool consists of a set of shell and Python scripts that provide AI driven feedback by using Large Language Model (LLM) prompts. The tool only supports OpenAI API keys which allows it to make calls to the OpenAI models.

The tool consists of a bash script (`feedback.sh`) that orchestrates the entire feedback process, a Python script (`query-ai.py`) that sends the information to the OpenAI API and returns the AI's response, and another Python script (`send-mail.py`) that sends the feedback report via email. The user runs the feedback script with a repository directory as input. The tool then analyzes the repository structure, extracts Java class declarations, and file sizes, and generates a feedback report. This report is output in Markdown format.

The tool uses a system of specialized prompts for each analysis. The prompts are essentially instructions for the models on what criteria to analyze the repository on. The prompts contain specific criteria for the output formatting to always create consistent feedback. The prompts don't just give criteria but focus more on generating useful instructive feedback. It emphasizes explaining problems rather than simply identifying them. The prompts are sent to OpenAI's GPT-4o model, which then analyzes the repository and generates the feedback.

In the end, the tool produces a feedback report that covers the following areas:

- **Repository contents** — evaluates whether essential elements such as source code, documentation, and tests are present.
- **Project architecture** — assesses the organization and modularity of the codebase and the clarity of its structure.
- **Source code style** — reviews consistency, readability, and adherence to coding conventions.

- **Project build specification** — checks if the tool can be reliably built and run using standard tools by examining common build configuration files.
- **Configuration management** — examines the use of version control, commit quality, and branching practices.

The generated feedback report is intended to help students understand their strengths and weaknesses when it comes to software engineering practices.

2.3 LUdev

LUDev is a virtual platform run by students at Leiden University that connects students with real-word tech projects [1]. It is a unique way for students to apply their theoretical knowledge to actual practical projects. Participating companies can sign up for LUDev and submit their project proposals, which are then matched to student teams. These projects typically span around 4-5 months and challenge students to develop these projects in a structured and methodological way. LUDev projects often emphasize collaborative development using industry standard tools such as GitHub, Git, CI/CD pipelines and Agile/Scrum methodologies.

This setting creates a perfect environment to evaluate educational tools that aim to improve software engineering best practices. As students are expected to work in teams, they must follow a structured way of developing. Since students are expected to manage their repositories, write clean and maintainable code, and document their work thoroughly, the quality of their work can be meaningfully assessed. Tools like ai-repo-feedback which provide feedback on repository structure, code style and documentation would thus be a perfect fit. It can help guide students in developing industry best practices.

2.4 AI powered peer review

Recent research has begun to explore the integration of generative AI in educational code review processes. One of these studies was conducted by Oliveira et al. [8] who investigated the use of generative AI to automate peer review processes for computer science students. The research provided a lot valuable insights that relate to the application of AI-generated feedback in programming education.

The research was carried out on 80 computer science students from the University of Melbourne. Oliveira et al. implemented an AI-powered peer review tool that uses OpenAI's GPT 3.5 model with integrated GitHub actions. The tool performed checklist based assessments to automatically evaluate student repositories. This approach is similar compared to this research; both leverage Large language models to provide automated feedback on student repositories.

The study revealed several important findings relevant to AI-generated feedback effectiveness. The AI powered peer review tool was able to increase student engagement by providing timely and accessible feedback. The participants noted that ChatGPT was able to provide feedback in seconds compared to an afternoon spent on peer review. This aspect of increased efficiency due to a faster response time aligns well with the expected potential benefits of automated feedback tools.

However, Oliveira et al. [8] also identified some very important limitations in AI generated feedback tools. While the AI system was able to identify a higher number of issues overall (average of 26,7 issues compared to 9.5 in peer reviews), human reviewers showed a superior ability to identify logical and structural issues. It shows that human reviewers have a deeper understanding of project context and requirements, something AI system often lack when analyzing code in isolation. So even though an AI system can identify more issues it shows a lack in quality and scope. Students noted that AI feedback could be "very generic" and struggled with providing specific, contextual recommendations students could actually act upon.

The study also found that students often went beyond the prescribed AI review process. They used ChatGPT conversationally to debug code and resolve issues. It suggests that AI feedback tools may be most effective when they provide continuous feedback in the form of a dialogue and exploration rather than providing a one-time assessment.

The research of Oliveira et al. showed both the potential and limitations of AI-generated code feedback in education. While their findings indicate that AI tools definitely show potential improvement in efficiency by timely identifying code issues. It also indicated some potential limitations in providing contextual, project-specific guidance compared to human based feedback. The study found that this lack of contextual guidance was often compensated for by students through continued conversational feedback with AI tools like ChatGPT. This research creates a foundation for understanding the potential benefits and challenges of AI-generated feedback in education, indicating points of improvement and focus.

This study show that AI feedback tools excel in speed and breadth but fall short in providing deeper insights. This gap narrows when AI is used in a interactive way, allowing students to refine and clarify feedback through dialogue rather than relying on a one-off assessment. Continuous AI interaction helps produce more relevant guidance which can compensate for its initial lack of depth. Future tools should combine AI's efficiency with human-like adaptibility which would offer the greatest potential for enchancing education code review.

2.5 AI-generated Programming Feedback

Alyoshyna [2] conducted a study at the University of Twente, examining the use of integrating AI generated feedback systems for programming exercises in higher education. The study uses a mixed-methods approach by combining a systematic literature review with an experimental evaluation involving student participants. Four different large language models (LLMs) –OpenAI GPT-3.5, Github Copilot, Google Gemini and a HuggingFace model– were used to evaluate their ability to generate useful feedback on programming exercises. The evaluation was done using established technology acceptance frameworks (UTAUT [6] and TAM [5]), that measure three key dimensions that align with the research objectives. The focus was on evaluating the perceived usefulness, perceived ease of use and behavioral intention to use.

The research revealed several key findings on students perception of AI generated feedback. Of the 4 different LLMs that were evaluated GitHub Copilot received the highest satisfaction ratings by a clear margin. With 4 out of 5 students rating it as most useful whilst only 1/5 rated the second best model, Google Gemini, as most useful. The study states that the high satisfaction rate was primarily due to GitHub Copilots structured approach, accuracy in identifying small mistakes and clear explanations. In contrast, the HuggingFace model was consistently rated as least useful across all participants because it lacked any actionable suggestions and provided overly general feedback. Overall, the students demonstrated a clear preference for feedback that was specific and actionable. This suggests that students valued clear explanations that helped them understand both the problems and the solutions, as well as structured presentation of the feedback points.

However, the study also showed several limitations in AI-generated feedback. Students reported issues with the accuracy of the feedback, where AI generated feedback sometimes provided incorrect or misleading suggestions. Another issue was the occurrence of generic feedback, with students reporting unhelpful feedback that was unable to address specific problems. Some AI systems tend to provide broad programming advice rather than advice on a specific exercise. There were also a few cases of circular reasoning and irrelevant content in AI responses, particularly with GPT-3.5 repeating the same points wihtout adding value.

Alyoshyna's work contributes to a growing body of research on AI generated feedback in education. The study provides valuable insights for educators considering the implementation of AI feedback systems. The findings suggest that AI-generated feedback shows a lot of promise in terms of scalability and timely reponses, the technology still requires refinement to match the contextual understanding of human provided feedback.

2.6 AIRSim: AI powered feedback tool

Leong and Sung introduced AIRSim (AI response simulator) [4], a tool that uses generative AI (ChatGPT) to create feedback data for students. It was mainly designed for the cafe and restaurant discipline so students can practice analyzing open ended customer survey responses. The tool addresses a common challenge in hospitality education: the lack of diverse and realistic customer feedback for training purposes. Educators can upload customized questionnaires on AIRSim for their students and define variables such as age, nationality and gender. The AI then simulates responses, generating a structured dataset in formats like Excel, which students then can use to analyze. This simulation based approach helps students learn by exposing them to a wide range of realistic feedback scenarios that would otherwise be hard to obtain.

To evaluate the effectivness of AIRSim, 16 experiments were conducted with varying number of questions and quantity of simulated participants. Entropy Index was used to measure the diversity of generated responses, demonstrating that the tool produced high variability datasets. The study highlighted how different levels of entropy affect the complexity and value of the learning experience. Higher entropy provided more challenging and realistic scenarios for student analysis. AIRSim contributes to the field of generative AI in education by innovatively integrating participants demographics for realism and export ready-to-analyze datasets. The tool reflects a broader trend in AI-enhanced education, where generative AI models are increasingly used not only for automating processes but also for fostering critical thinking among students.

2.7 Conclusion

These three studies were selected for related work because they examine generative AI applications in educational context. They revealed both some potential and limitations in feedback generated by AI. All studies showed that AI tools excel in efficiency and scalability but struggle with the depth and quality of the feedback. They showed a tendency toward generic responses where speed comes at the expense of the depth of the feedback. Together the studies demonstrate that effective AI implementation in education requires balancing computational advantages with human like contextual understanding.

Chapter 3

Method

3.1 Overview

The study follows a structured process consisting of three main phases: (1) development and implementation of the tool, (2) feedback distribution and (3) evaluation through a survey. This structure was chosen because it forms a perfect cycle to develop the tool, apply the tool and assess its impact from the student perspective.

The development and implementation phase focuses on modifying and expanding the original AI Repo Feedback tool to also support Python-based repositories in addition to Java. This phase directly addressed *Support RQ.1*, as it explored how the tool could be adapted for LUDev student projects.

The feedback distribution and survey phase focused on developing the survey and setting up the conditions to evaluate the feedback tools usefulness and impact. This phase is linked to *Support RQ.2* and the main research question.

Finally, the evaluation phase consisted of collecting survey reponses to evaluate the clarity, usefulness and actionability of the feedback. The student responses provided valuable insights into *Support RQ.2* (usefulness and clarity of the feedback) and *Support RQ.3* (strengths, weaknesses and suggestions for improvements). These insights from the supporting questions provided evidence to answer the main research question about overall effectiveness.

3.2 Development and implementation phase

The original tool was developed to provide feedback on Java projects. Since a lot of the student projects on the LUDev platform were Python based, the first step of this research was to expand the tool to handle Python repositories. We started off with analyzing the tool so to determine which parts needed readjusting. The tool consisted of specific prompts and code that were are tailored to provide feedback on Java code only. All the code was meant to search for Java files and analyze based on Java specific characteristics. After determining the key components to readjust it became clear what to focus on. The expanded tool needed to have Python specific prompts, handle Python specific features and provide feedback on Python best practices.

For example, the original tool would look for Java files using the command '`find .-name\*.java`'. In contrast Python projects often use a different structure, with init.py files to define packages and a focus on module organization. The new expanded tool uses the command '`find .-name\*.py`' to locate Python files.

After step by step going through the code and readjusting and adding where needed, the new tool was able to handle both Python and Java projects. The first component that changed was the language detection part so that the tool could handle Python code. The script now goes through a repository by looking for both Python and Java files. After being able to detect Python code it now needed to be able to analyze the code so we added some Python specific functions. Three functions were added to analyze Python files on the module structure and architecture, source code files, and provide feedback on individual Python files. The script is also able to handle some specific Python files such as init files and requirement files.

The next important component that was improved were the specific prompts that handle feedback generation and correct output formatting. The new prompts were tailored to analyze Python specific features (classes, functions, module organization and package structures, Python specific best practices (PEP 8 [11]), dependencies and requirements management). PEP 8 is the official style guide for Python code and provides conventions for code layout, naming, and other formatting details. Adhering to PEP 8 helps ensure that Python code is readable and consistent. The PEP8 guidelines are easy to implement in the prompts of this tool. They are well documented and easy to reference in prompts for LLMs. There are also other ways to check for industry best practice compliance but they are mainly tools that would essentially create a separate feedback pipeline outside the original tools GPT model. Creating a system with 2 separate tools was out of the scope of this research. Tools like Flake8 also give raw error code and line numbers that aren't well formatted for feedback reports.

The feedback reports were generated by first creating a local temporary repository folder. In that folder all the student repositories were cloned so it could be used to generate feedback on. A small script was created to run the feedback shell of the ai-repo-feedback tool on every repositories in the folder. The script would run the feedback script on each repository sequentially, save the generated feedback report in another folder and delete the student repository.

Due to the original tool's reliance on hard-coded prompts that were designed for a specific programming language, developing a tool capable of handling an arbitrary number of coding languages was beyond the scope of this research. Creating such a tool would require building from scratch and designing versatile prompts that can effectively work across any programming language. These prompts would need to be high quality and incorporate the numerous rules and conventions specific to each programming language.

3.3 Feedback distribution and survey phase

To assess the usefulness of the adjusted tool and its effectiveness, the tool needed to be tested on student repositories. In collaboration with the lecturers of the course Software engineering, we gained access to the LUDev student repositories of the current academic year 2024/2025. With this access we generated feedback reports using the tool of every single student repository that contained mainly Python or Java code. The generated feedback reports together with a survey were distributed to the respective students using their student emails for evaluation.

The survey is designed to evaluate the AI-generated feedback reports generated from the LUDev student software engineering projects. The survey aims to answer some of the research questions about usefulness, the effectiveness, and strengths and weaknesses of the AI Repo Feedback tool. Determining the usefulness of the tool means to understand if the feedback was clear, relevant and of any value. Determining the effectiviness means to understand if the feedback actually lead to any improvements or learning outcomes. The survey (see Survey Link) is segmented into 4 sections to help analyze responses and provide a clear structure to the response. The purpose of the first section is to control variables for analysis by asking for contextual information. The second part is about the usefulness of the feedback report, focusing on clarity, perceived usefulness and most valuable sections. The third section is about the effectiveness of the report by assessing whether the students actually applied the feedback, the time they spent engaging with the feedback, and the perceived learning. The final section is about overall evaluation to gather broader opinions about satisfaction and future use. In the last part there is also a comparison between the AI generated feedback and traditional feedback. This survey is a short but focused way to find answers to the research questions due to the targeted and concise questions stated.

3.4 Evaluation phase

This study focused on evaluating the clarity, usefulness and actionability of the feedback reports generated by the AI Repo Feedback tool. Success was defined as feedback that was clear for the students, relevant to their projects and helpful in improving their software engineering practices.

The survey had both quantitative and qualitative questions to capture these aspects. The Likert scale questions were used to quantitatively assess the students opinion on the clarity, usefulness, and satisfaction in a structured way. The open-ended question was qualitative because it provided deeper insights into which parts of the feedback could use some more improving by asking about the respondents experience.

The results of the survey were analyzed by aggregating and visualizing the quantitative data, and by grouping the qualitative answers to identify recurring strengths, weaknesses and areas for improvement. The analysis was used to evaluate the effectiveness of the tool and to answer the research questions.

Chapter 4

Results

4.1 Feedback report generation

In total, 18 LUDev student repositories were selected for analysis using the new upgraded AI Repo Feedback tool. However, not all of these repositories contained enough Java or Python code to generate meaningful feedback reports. Of the 18 only 11 had enough Java or Python code in their projects, where 3 projects only contained 30% Python code. The other 8 had more than 70% of Java or Python code so they were able to provide enough data to create a comprehensive and useful feedback report. The 3 projects that had less than 30% of Python code were able to provide some feedback on only some small parts of the project that were Python. Only projects containing more than 30% Python code were used to ensure comprehensive feedback reports. While projects with less than 30% Python code could have more total lines of Python code, the limited Python content would result in incomplete feedback reports that do not adequately represent the Python-specific aspects of the code based. In the end 11 feedback reports were created using the enhanced ai-repo-feedback tool.

Below are several examples from different parts of the generated feedback reports. Listing 4.1 shows an example of feedback related to documentation and doc strings. Listing 4.2 shows the feedback given on the writing style of the Python code using the PEP 8 guide. Finally, Listing 4.3 shows some example suggestions each feedback report gives at the end of each feedback report.

~~~
### Documentation and Docstrings
- While the module has an introductory docstring, consider providing more
    detailed docstrings for functions and main logic sections. This enhances
    understanding of what each part of the code is doing:
  ```python
  def split_name(full_name):
      """
      Splits the full name into given, patronym, and surname.

      Parameters:
      full_name (str): The full name to split.

      Returns:
      tuple: (given, patronym, surname)
      """
  ```
~~~

Listing 4.1: Example feedback on documentation

```
### PEP 8 Compliance
The code mostly follows PEP 8 conventions, but here are a couple of
    suggestions:
- Add spaces around operators (e.g., ‘if distance < smallest_distance‘ could
    be more consistently formatted).
- Ensure all function names use the ‘snake_case‘ style as prescribed by PEP
    8  ‘correct_givenname‘ is consistent, but follow this for all function
    definitions.
```

Listing 4.2: Example feedback on official style guide for writing Python code

```
## Suggestions for Improvement

1. **Merge Message Consistency**: For merge commits, maintaining a clear and
    consistent message that describes what changes are being brought together
    can help all team members understand the changes made.
2. **Encourage Wider Participation**: Consider implementing practices that
    encourage all team members to contribute more frequently, such as pair
    programming or scheduled contributions.
3. **Refine Commit Messages**: Emphasizing the imperative mood across all
    commit messages uniformly would enhance clarity. Aim for every message to
    start with an action verb, e.g., "Fix", "Update", "Create".

Overall, the contributions are commendable, and with slight adjustments in
    practices, the team can achieve even greater coherence and collaboration in
    future projects.
```

Listing 4.3: Examples of given sugestions

4.2 Survey results

The survey was distributed to the 11 different project groups together with each respective feedback report. Initial responses were very low, so a reminder was sent out by the LUDev board to encourage responses. Paired with the reminder email, a reminder chat message was send to the whatsapp group chat of the current academic year who were taking part of the LUDev projects. The chat message helped a bit with increasing the response rate to a final total of 13 responses.

Although the survey was distributed to all groups and several reminders were sent out, the total number of responses collected stayed very low. As such, the findings of this research should be interpreted as exploratory insights that could be used for future work rather than a concrete statistically supported conclusion.

4.2.1 Contextual information

The survey started off with some questions to create some context for the responses in section 1. The findings were as follows: of the 11 responses, 6 of them were part of the same group as the author of this thesis, while the rest were individual responses from distinct project groups. Of these groups, almost half of the repositories consisted of mainly Python code (see Figure 4.1), with others being mainly Java, JavaScript or CSS. Based on the survey results, before receiving the feedback reports 45% of the respondents were already somewhat familiar with industry coding standards (e.g. modularity, documentation, PEP8) (See Figure 4.2). Even 20% of the respondents were *very familiar* with the industry standards. The other 25% were *not at all* familiar and the remaining 10% were *somewhat familiar.*

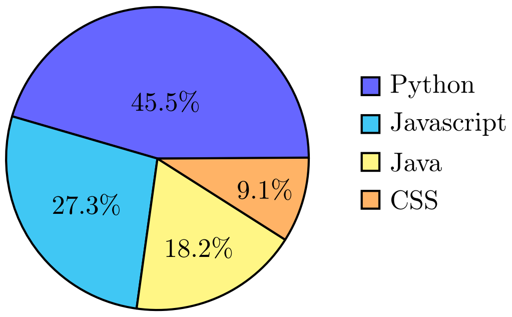

Figure 4.1: Primary programming language used in project

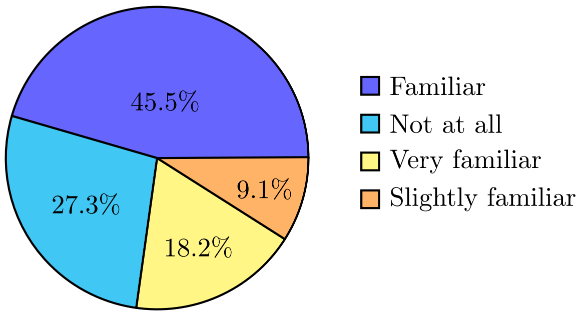

Figure 4.2: Familiarity with industry coding standards

4.2.2 Usefulness of feedback

Section 2 included four questions that were designed to evaluate the usefulness of the feedback. The first was on the general clarity of the general feedback, which resulted in almost half of the respondents giving it a score of 4 out of 5 (see Figure 4.3). The second question was about how well the feedback was able to identify areas of improvement, with more than half of the responses leaning towards very useful (see Figure 4.4). The third question was to determine how well the feedback was able to balance the praise and criticism in the feedback, with almost half of the responses falling in the middle, neither too critical nor too generic (see Figure 4.5). The last question was to determine how valuable some parts of the feedback report were. There were 5 sub questions that each asked the value of a specific part of the feedback report. The results of these 5 questions are compressed into Figure 4.6. The figure shows the number of responses on the y-axis and the five different sections on the x-axis. Each section on the x-axis is grouped together with different colored bars showing the score each section received from the respondents. The legend below the figure shows what each colored bar represents in terms of score.

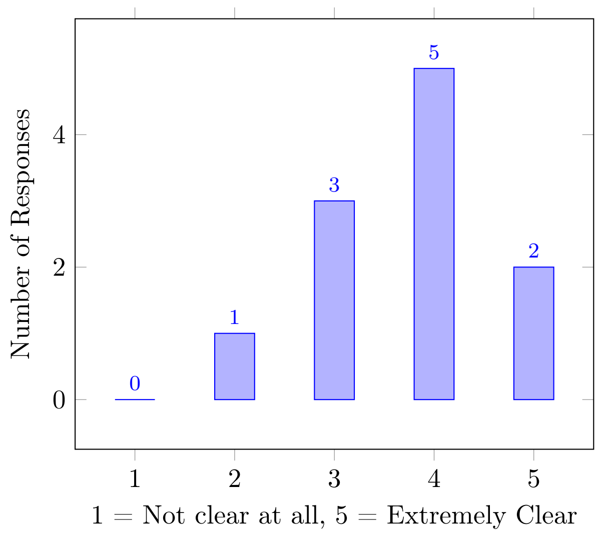

Figure 4.3: How clear was the feedback?

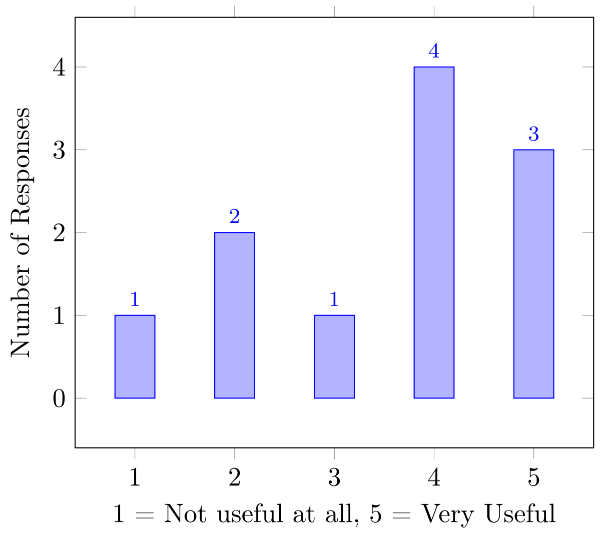

Figure 4.4: How useful was the feedback in identifying areas of improvement?

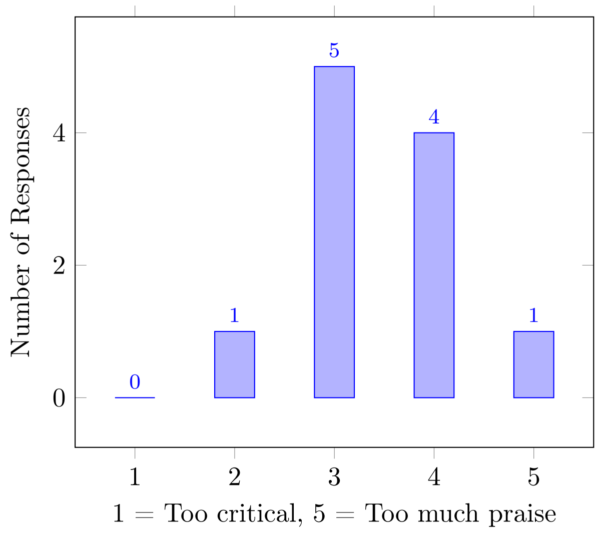

Figure 4.5: How would you rate the balance between praise and criticism in the feedback?

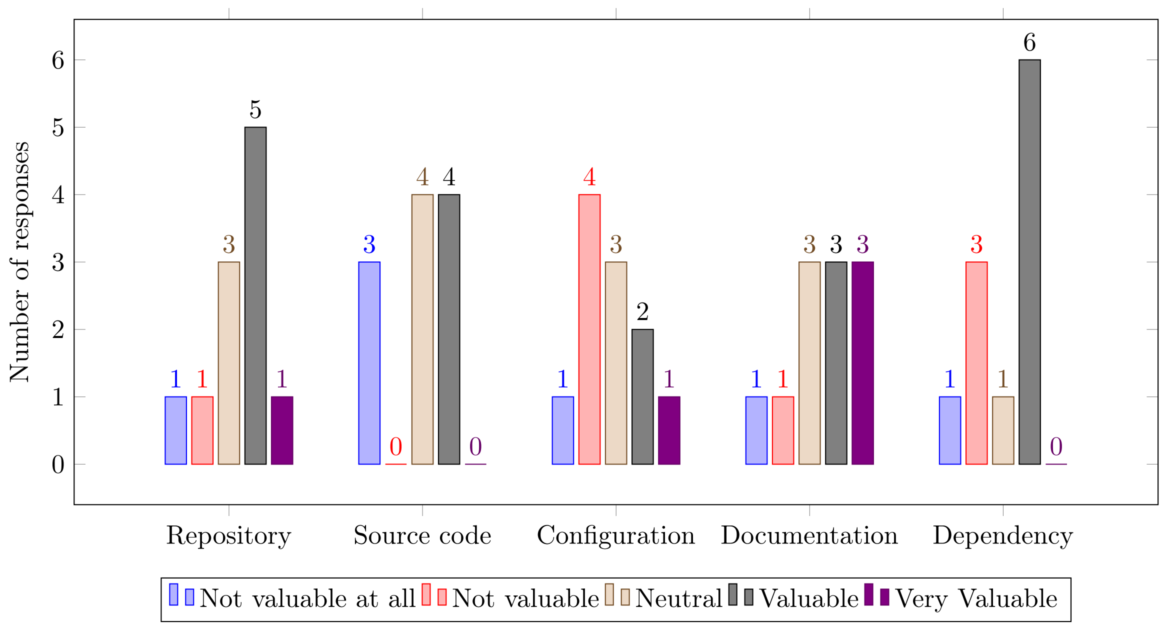

Figure 4.6: Distribution of responses on the 5 different sections of the feedback report

4.2.3 Effectiveness & Actionability

Section 3 of the survey included four questions that aimed to evaluate the effectiveness & actionability of the feedback. To create additional context, the first question (see Figure 4.9) in this section was about how much time was spent reviewing the feedback. With 80% of the respondents spending 3-5 minutes or more reviewing the feedback reports.

Then we asked if the respondents actually acted upon the suggestions that were provided in the feedback report. You can see in Figure 4.7) that half of the participants responded with partially, meaning they acted upon some suggestions that were provided. There was not one participant that answered with a *yes* to this question.

Figure 4.8 are the results of asking the participants if they feel more confident in improving their repository, with 10 out of the 11 participants answering it with a 3 or 4. The last question in this section aimed to determine if the feedback actually helped students understand the software engineering best practices better (see Figure 4.10). As can be seen in the diagram responses are spread out over the Likert scale, ranging from *No it was confusing* to *Yes significantly*.

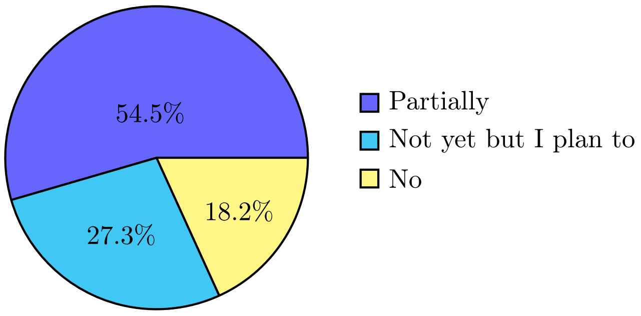

Figure 4.7: Action taken on suggestions

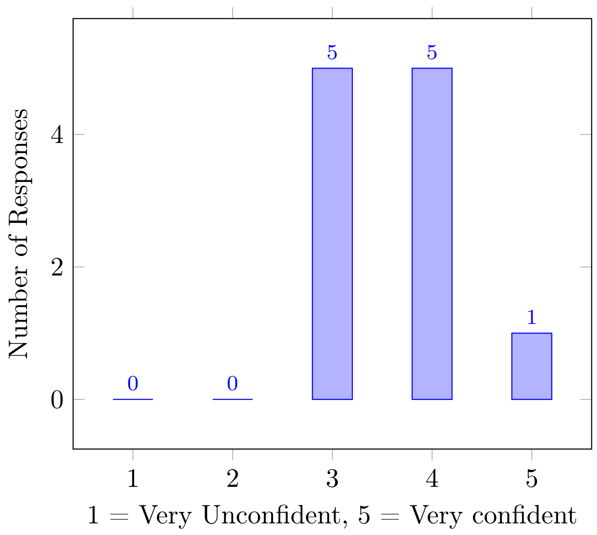

Figure 4.8: Confidence in improving repository

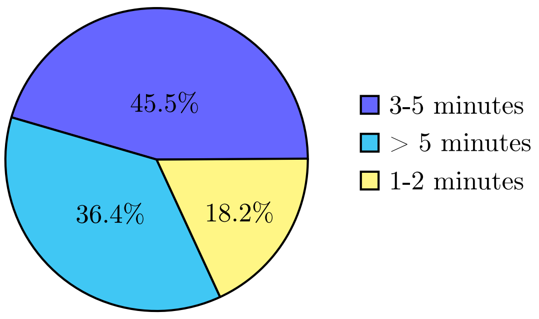

Figure 4.9: Time spent reviewing feedback

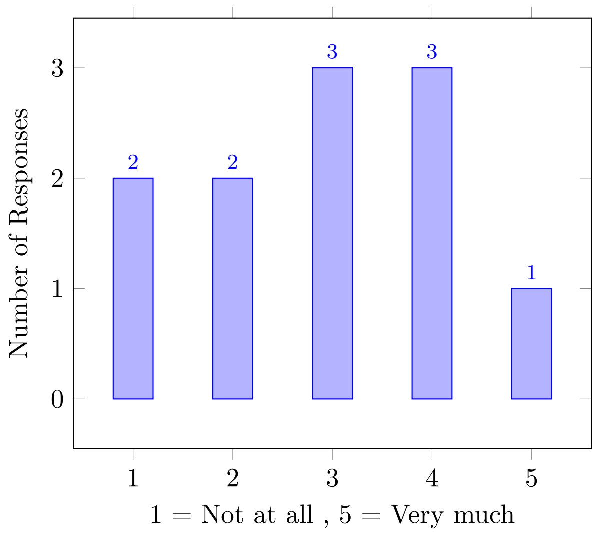

Figure 4.10: Better understanding of software engineering best practices

4.2.4 Overall evaluation

In section 4 of the survey the goal was to now evaluate the AI-Generated feedback report in general. There were 5 questions in this section which aimed to make the survey round and provide enough insights to answer the main research question.

The first question measured whether the participants would actually use the tool for future academic projects. The responses (see Figure 4.11) were split equally between whether they would use, might use, or would not, use the tool in the future.

The second questions was to help determine if the feedback actually helped the students become a better software developer. More than half of the participants (see Figure 4.12) felt that feedback helped them somewhat grow as a developer.

The third question in this section aimed at comparing the AI-Generated feedback with traditional feedback, to see what the students preferred. In Figure 4.13 you can see that half of the participants thought that the feedback generated by AI was worse than traditional feedback by teachers. Only 20% of the participants thought that the AI generated feedback was better than traditional feedback.

The fourth question builds on the previous question by assessing how the participants would use the AI-generated feedback compared to traditional feedback. In Figure 4.2.4 you can clearly see that the majority of participants would use the AI generated feedback *in addition to* traditional teacher feedback.

The last question in this section was to assess the final overall satisfaction of the participants with the AI generated feedback report. Figure 4.14 shows that half of the participants were quite satisfied with the feedback by giving it a 4.

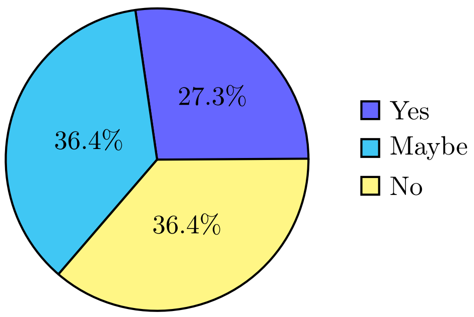

Figure 4.11: Would you use in the future

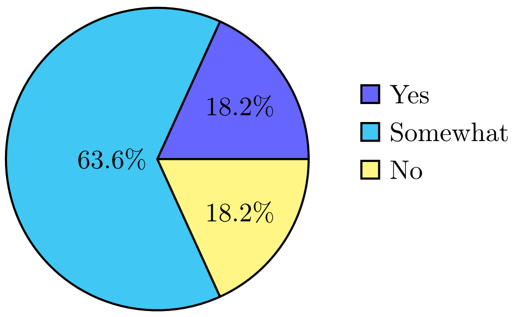

Figure 4.12: Help grow as a software developer

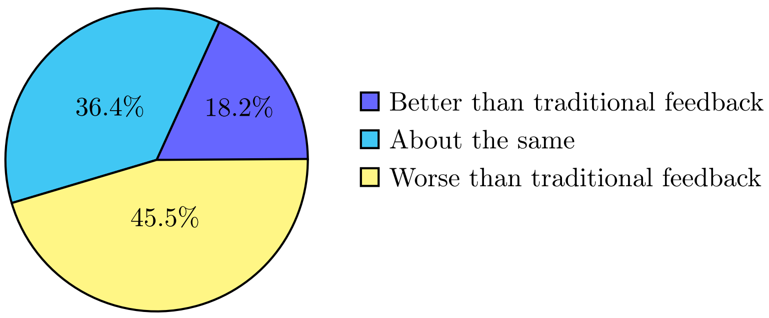

Figure 4.13: AI Generated feedback vs Traditional Feedback

AI-Generated feedback in addition

or instead of teacher feedback

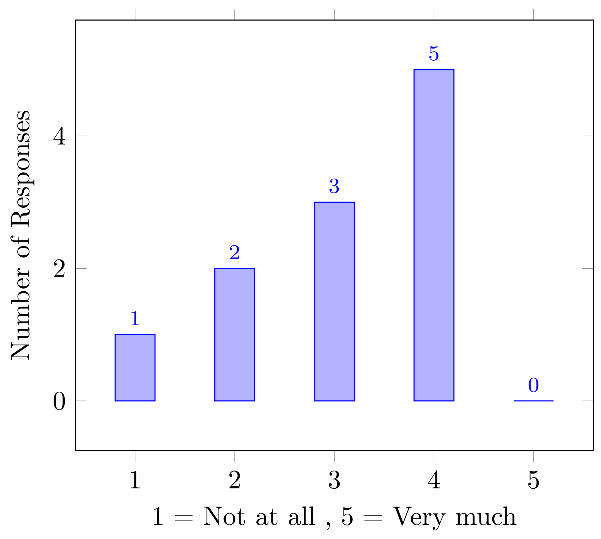

Figure 4.14: Overall satisfaction with the feedback report

Chapter 5

Discussion

5.1 Results in context

The results of this study suggest that AI Generated feedback holds notable promise in enhancing programming education, due to the high overall satisfaction rate received in the last question of the survey. However, given the low response rate this research takes a more exploratory perspective on the perceived effectiveness of AI-generated feedback. Rather than a robust statistical conclusion this research provides insights that offer valuable directions for future research. With this in mind, the following subsections offer an interpretation of the findings of each section in the survey.

5.1.1 Section 1 of the survey

The first few questions in the survey helped establish the context and the scope of the responses. From the results we see that half of the responses were from projects that contained primarily Python code (see Figure 4.1), and approximately half reported being already familiar with industry coding standards (see Figure 4.2).

5.1.2 Section 2 of the survey

The second section in the survey, which focused on the usefulness of the feedback, indicated that participants generally found the feedback clear (see Figure 4.4). Responses on how well the feedback was able to identify areas of improvement in the project were mixed, indicating that some students found the feedback to be very helpful in helping to identify areas of improvement while others found it not useful at all. This could be due to the varying levels of the student experience with coding standards. Less experienced students likely found the feedback more beneficial, as the AI generated feedback was able to identify more issues in their projects. In contrast, student with more experience may have produced code with fewer detectable issues, resulting in less impactful feedback.

The survey also asked participants to evaluate the value of five distinct parts of the AI-generated feedback report. The results reveal interesting differences in how the students perceived the usefulness of different parts of the feedback (see Figure 4.6).

The feedback on Source code and Dependency received the highest proportion of positive ratings, with most of the responses marking them as *valuable* or *very valuable.* This suggest that students particularly appreciated insights related to code style structure and dependency management. Areas that are crucial for code maintainability and reproducibility.

The repository section in the report that focused on the the overall project structure and completeness had a relatively even spread across all value categories. This may indicate that students had differing expectations or levels of understanding regarding repository best practices.

In contrast, the feedback on configuration and documentation saw a more varied distribution. Some students found these parts of the feedback valuable while other rated them as neutral or even invaluable. The mixed responses may reflect varying project maturity levels or differences in how much documentation was emphasized during development.

Overall, the results on the value of different parts of the feedback suggest that students value detailed and technical feedback (e.g. on source code and dependencies) a lot more than broader structural or organizational guidance.

5.1.3 Section 3 of the survey

This section focused on evaluating the effectivness & actionability of the feedback. The first notable find of importance for context from the results of this section is that the participants spend a decent amount of time reviewing and understanding the feedback. 45.5% of the participants voted that they spend 3-5 minutes reviewing the feedback, with another 36.4% spending more than 5 minutes (see Figure 4.9). This indicates that the participants took their time to evaluate the feedback which gives the findings greater credibility.

Another important find is that half of the participants stated that they acted (partially) upon the suggestions provided in the feedback report, while the other half stated that they did not (yet) act upon the suggestions (see Figure 4.7). This highlights a mixed perception of the actionability of the feedback, suggesting that some students found the suggestions implementable, some found them not valuable enough to apply. Interestingly, another question in this section revealed that all participants rated their confidence in improving their repository above a 3 on a 5-point scale. This indicates that while some students felt that the feedback could improve their repository, it was not enough to actually apply it. I was not able to see any clear incorporation of the feedback through the commits on GitHub. I compared the feedback reports with the commits but was not able to see concrete pieces of feedback being integrated into the code. There were also no clear commit messages stating that the changes were based on the received feedback.

For the last question in this section the results in Figure 4.10 show a broad distribution of responses when asked if the feedback helped with understanding software engineering best practices. While some participants felt that the feedback helped them grow significantly, others found it not useful at all. The majority, however, experienced only moderate gains in enhancing their understanding of software engineering best practices, indicating room for improvement in the depth of the feedback.

5.1.4 Section 4 of the survey

The final section of the survey aimed to assess the overall perception of the AI-generated feedback and its place in the academic world. Figure 4.11 shows that the responses were almost equally split between whether they would use the tool again, if they were unsure, or would not use the tool again. Indicating that some students saw clear use and value in the feedback, many remained undecided or convinced of its utility. Figure 4.12 further supports this by showing that only 18.2% believed that the feedback directly contributed to their growth as a software developer. With the majority (63.6%) acknowledging that it partially helped them grow as a developer.

The comparison with traditional feedback, presented in Figure 4.13, revealed that almost half of the participants found AI-generated feedback worse than teacher feedback. Only 18.2% rated the AI generated feedback as better than traditional teacher feedback. However, despite the clear preference for human evaluation, Figure 4.2.4 shows that 81.8% of students would still be open to using AI-generated feedback alongside teacher feedback, suggesting a strong potential for a blended approach rather than replacement.

Finally, Figure 4.14 shows that general satisfaction with the overall feedback report was moderate, with most students rating their satisfaction between 3 and 4 on a 5-point scale. This reflects the general acceptance of AI-generated feedback but also points to the need for refinement in making the feedback more impactful.

5.2 Limitations

There are several limitations that must be acknowledged when interpreting these findings:

5.2.1 Generalizability

The results of this research provides valuable insights into the usefulness of AI generated feedback on student code. However, there are several limitations that constrain the generalizability and statistical reliability of the results. Firstly, the study experienced a low response rate, with only a few students completing the questionnaire. This limited sample size implies that the results cannot be statistically analyzed which means that the results should be interpreted as exploratory rather than conclusive. Secondly, this research was conducted at only a single academic institution, Leiden University, and focused only on a small portion of students participating in the LUDev software engineering platforms. As a result, the students perception of the feedback and their criteria for evaluating it depends on the specific structure of the curriculum and teaching style. This limits the transferability of the findings to other educational institutions with different curriculum and teaching styles.

5.2.2 Affiliation bias

Due to half of the responses coming from my own project group it introduces the affiliation bias. It means that the responses can be unconsciously or consciously more favorable or lenient because of the respondents awareness of my role in this research. This can lead to biased responses that could inflate ratings of the feedback's usefulness or effectiveness.

5.2.3 Subjective nature of usefulness

Usefulness and clarity are inherently subjective qualities. Different students may interpret the same feedback in various ways, influenced by their own personal level of experience, confidence or expectations. To mitigate this issue of different perceptual differences, we used Likert scale question. This approach helps reduce the problem of different individual scoring tendencies. For example, while one student might rate feedback as a 6 out of 10 when they perceive it as "good", another might rate the same experience a 7. A Likert scale simplifies this by grouping responses into more interpretable categories that are less sensitive to individual scoring nuances.

5.2.4 Prompt engineering

The effectiveness of these feedback reports also relies heavily on prompt engineering. Poorly designed prompts may lead to vague generic or irrelevant feedback which could affect the student perceptions. Limited expertise in prompt engineering could lead to suboptimal prompts, which in turn leads to low quality feedback. Therefore, the actual effectiveness of AI-generated feedback is strongly tied to the quality of the prompts used. However, evaluating and optimizing prompts were beyond the scope of this research. As a results, the value of the outcomes depends heavily on the prompts used, which were not systematically assessed.

5.2.5 Self selection bias

As the evaluation of the tool's effectiveness relied on a survey, both survey bias and response rates pose significant limitations. The data would have been more robust if full class participation had been achieved, yet in practice only a small number of students responded. This introduces the risk of self selection bias, where students with particularly strong opinions (positive or negative) are most likely to respond, potentially skewing the results.

Chapter 6

Conclusion

This research set out to explore the potential of AI generated feedback to support students in software engineering education. By adapting and expanding the AI Repo Feedback tool developed by Diomidis Spinellis, the study was able to evaluate both Python and Java repositories. A crucial step due to the diverse coding languages used in LUDev student projects. The tool successfully generated feedback reports from real world student projects which were then evaluated using a survey.

This research contributes to a new growing body of research on AI generated feedback in two ways. Firstly, the evaluation phase of the research provides exploratory empirical insights into the clarity, usefulness, and actionability of AI generated feedback for student coding projects. While results indicate that students found the feedback clear and somewhat useful. Only some reportedly acted on the feedback indicating a mixed perceived actionability. There was also a mixed willingness to use the tool again. Overall, the majority found the feedback somewhat helpful, suggesting that it is best suited as a supplementary tool, with general satisfaction being moderate. Secondly, the first part of this research demonstrates the feasibility of integrating generative AI into educational feedback processes. The enhanced tool was successfully extended to also support Python projects, incorporating language specific prompts to assess adherence to industry best practices.

Ultimately this research highlights both the potential and limitations of AI generated feedback in education. It offers exploratory insights that can guide future work in this emerging field.

6.1 Answers to the research questions

Main Research Question:*How effective is the AI Repo Feedback tool in providing meaningful and actionable feedback on student software engineering projects?*

The results indicate that the AI generated feedback is moderately effective in providing both meaningful and actionable feedback. Many students found the feedback to be clear and helpful, but viewed it more as a supplementary tool rather than a replacement for traditional feedback.

Support RQ.1*How can the AI Repo Feedback tool be modified and extended to support Python-based repositories?*

The tool was succesfully adapted to analyze Python based projects by modifying the underlying script, prompts and language detection logic. Python specific feedback mechanisms were added, including checks for moduling structure, code style (PEP8), and dependency management.

Support RQ.2*How useful and clear do LUDev students find AI-generated feedback and which aspect of the feedback report is considered most valuable?*

Most students deemed the AI generated feedback to be clear and moderately useful. Survey results suggest that students less familiar with industry coding standards responded more positively to the AI-generated feedback than those with more experience. Specifically, students who were "Not at all" (4 students) or "Slightly familiar" (2 students) with standards like PEP8 rated the feedback higher in clarity and usefulness, while those who were "Very familiar" (2 students) gave lower ratings and found the feedback too generic or even misleading. While the sample size is small and biased, the trend indicates that the tool was especially helpful for less experienced students by highlighting issues they may not yet recognize on their own.

The most valued parts of the feedback were those that focused on source code and dependency handling, while configuration and documentation feedback received mixed responses. That is because source code and dependency handling feedback was more specific, pointing to exact lines or violations like PEP 8 issues, making it clearer and easier to act on. Feedback on configuration and documentation would often be more broad and vague, such as "add more details" or "missing these files". As a result, students valued the more targeted feedback on source code and dependency handling more.

Support RQ.3 *What are the strengths and weaknesses of AI-generated feedback in software engineering education and how can the tool be improved for future use?*

The strengths of this tool lies in its ability to provide timely feedback at scale when identifying code issues. Weaknesses includes the tendency to offer generic suggestions, as students value direct technical suggestions. Improvements could include refining prompt design, expanding the tool to dynamically analyze different programming languages, and creating a system for iterative or dialog based feedback.

6.2 Future work

6.2.1 Dynamic code detection

Currently, the expanded tool works only on Java and Python. Initial hopes in this research were to expand the tool so it could dynamically detect any programming language and generate necessary the feedback. Due to time constraints this research was limited to only expanding the tool to include Python projects. For now the feedback tool simply iterates through a repository searching for Java and Python file, then send hard-coded prompts to a model to generate feedback. For future work, the tool could be modified to support any coding language by incorporating dynamic language detection and smart prompting generation tailored to the syntax and conventions of the detected language.

6.2.2 Compare with human feedback

A valuable direction for future work could be to compare AI generated feedback with traditional human feedback. The findings in this research revealed that generally the students found the AI generated feedback clear and somewhat useful, yet nearly half of the students considered the AI feedback to be inferior to human feedback. This suggests that although AI feedback offers scalable and timely responses, it lacks the contextual understanding, nuance and adaptability humans. A controlled study comparing both feedback types would provide more concrete insights into the relative strengths and weaknesses of each. The results of such research would be able to enhance the quality of AI feedback tool and how to best integrate it into educational workflows.

6.2.3 Continuous feedback

A valuable direction for future work is to enhance the tool to also incorporate continuous feedback throughout the project life-cycle. The current tool only provides feedback one time, which limits students' ability to act on suggestions and refine their work iteratively. This limitation was reflected in the results, where only half of the participants reported applying just some of the feedback provided. Additionally, while students felt confident in improving their project code, the lack of follow up suggestions reduced the overall actionability of the tool.

Implementing a feedback loop where students can resubmit improved versions could support incremental learning. The new tool should have the ability to provide context aware feedback, recognizing which parts of the code have already seen changes, so that new suggestions are relevant and build upon prior changes. Such a system would improve the actionability and enhance the education value of the tool. Moreover, it could also help instructors monitor student progress by increasing transparencies of the development process.

6.2.4 Explainable AI

Another valuable direction for future work is to incorporate Explainable AI (XAI) techniques into the feedback system. The current tool does not explain why certain issues were flagged or how recommendations were derived. This lack of transparency was reflected in the results: many participants found the feedback to be too generic and not sufficiently contextualized to their specific project. Incorporating XAI techniques such as linking feedback to specific code lines, referencing relevant coding standards, or describing the model's reasoning, would help clarify and justify the feedback process. By making the decision making process more transparent, students can better understand the rationale behind suggestions. This would lead to higher actionability from the students and also improve their learning process.

Bibliography

[1] Leiden university development team (LUDev), 2025. URL https://ludev.nl/. Accessed: April 28, 2025.

[2] Yuliya Alyoshyna. in programming education: Automated feedback systems for personalized learning. Master's thesis, University of Twente, Enschede, The Netherlands, July 2024. TScIT 41, July 7, 2024.

[3] GitHub. Github copilot. https://github.com/features/copilot, 2025. Accessed: August 7, 2025.

[4] Kelvin Leong and Anna Sung. Introducing AIRSim: An innovative AI-driven feedback generation tool for supporting student learning. *Technology, Knowledge and Learning*, 2025. doi: 10.1007/s10758-025-09835-9. URL https://doi.org/10.1007/s10758-025-09835-9.

[5] David Marikyan and Savvas Papagiannidis. Technology acceptance model: A review. In S. Papagiannidis (Ed.), *TheoryHub Book*, Newcastle University, 2025. URL https://open.ncl.ac.uk/theories/1/technology-acceptance-model/. Accessed: 2025-08-10.

[6] David Marikyan and Savvas Papagiannidis. Unified theory of acceptance and use of technology: A review. In S. Papagiannidis (Ed.), *TheoryHub Book*, Newcastle University, 2025. URL https://open.ncl.ac.uk/theories/2/unified-theory-of-acceptance-and-use-of-technology/. Accessed: 2025-08-10.

[7] Jaromir Savelka Paul Denny Mark Liffiton, Brad Sheese. CodeHelp: Using large language models with guardrails for scalable support in programming classes, August 2023. URL https://arxiv.org/abs/2308.06921. Accessed: 2025-06-13.

[8] Eduardo Araujo Oliveira, Shannon Rios, and Zhuoxuan Jiang. AI-powered peer review process: An approach to enhance computer science students' engagement with code review in industry-based subjects. In *ASCILITE 2023: People, Partnerships and Pedagogies*, pages 184–194. ASCILITE, 2023. doi: 10.14742/apubs.2023.482. URL https://www.researchgate.net/publication/376205465.

[9] V. Saravanan, S. Kavitha, S. Ravi, A. Seetha, Ch. Rambabu, and Tatiraju V. Rajani Kanth. Generative AI in software engineering: Revolutionizing code generation and debugging. *International Journal of Computational and Experimental Science and Engineering (IJCESEN)*, 11(2):2908–2917, 2025. doi: 10.22399/ijcesen.1718. URL https://doi.org/10.22399/ijcesen.1718.

[10] Diomidis Spinellis. ai-repo-feedback: AI-based repository feedback. https://github.com/dspinellis/ai-repo-feedback, 2023. Accessed: April 28, 2025.

[11] Guido van Rossum, Barry Warsaw, and Nick Coghlan. PEP 8 – style guide for Python code. https://peps.python.org/pep-0008/, 2001. Accessed: [Insert your access date here].

.1 Survey Link

The full survey used in this study can be accessed online via the following link: https://docs.google.com/forms/d/e/1FAIpQLSeZhY_tWM5DVZbOsUzDkKquoM-ZqT1mux9OqXkLFP8j9ZasKA/viewform?usp=header